Towards a generic compilation approach for quantum circuits through resynthesis

Arianne Meijer - van de Griend

***Abstract*—In this paper, we propose a generic quantum circuit resynthesis approach for compilation. We use an intermediate representation consisting of Paulistrings over $\{Z, I\}$ and $\{X, I\}$ called a "mixed ZX-phase polynomial". From this universal representation, we generate a completely new circuit such that all multi-qubit gates (CNOTs) are satisfying a given quantum architecture. Moreover, we attempt to minimize the amount of generated gates.**

We show that for large circuits, containing ≥ 100 Paulistrings, our algorithm generates significantly less CNOTs than previous methods, as well as both the TKET compiler and Qiskit transpiler.

Note for QPL reviewers: when implementing PermRowCol, I realized that my implementation of the paritysynth [24] did not properly correspond to the pseudocode and I have changed it. As a result, the algorithm is much slower, about 2 minutes on average to compile a 20 qubit circuit with 100 gadgets for the target devices. This means that running the script that compares the different methods took approximately 37 hours on a MacBook Air M1 2020. Because this is on the brink of what is acceptable runtime, I have implemented my previous algorithm [16] to compare against. It is much faster but less CNOT-efficient than paritysynth (see Tables I and II, and also [24]).
I have rerun the experiments and updated the tables with my latest results (April 12th) which also include the runtimes. I am currently rewriting the text to fit these new findings, but the methods should be clear for those familiar with the component algorithms ([16, 24, 9, 12]) after reading the caption of Table I

I. Introduction

Small scale quantum computing has recently become accessible to the public through cloud APIs like AWS Braket [2], IBM Qiskit [1], among others. With these new computing platforms comes the need for new compilation methods for quantum programs. Although some classical compilation methods can still be used for quantum computers, some compilation problems require a new solution due to the quantum nature of these devices. For example, most devices do not allow operation between arbitrary qubits. This is generally true of classical computers as well, but there we can solve the problem by copying bits. However, copying qubits is not possible due to the no-cloning theorem [27]. Similarly, optimizing quantum programs is a complex task. We somehow need to understand what the program is doing without accidentally executing it. Because simulating arbitrary quantum programs is infeasible on classical computers. In this paper, we propose a new compilation method for quantum circuits that optimizes the quantum circuit, and in doing so, it generates only quantum operations as allowed by a target quantum computer.

We essentially do this by throwing away the original circuit and generating a completely new circuit. This means that we do not need to cut the circuit into small pieces (as in e.g. [8, 17, 11]) and we do not generate any SWAP gates explicitly (as in [12, 25, 18]). Hence, this is a full-stack re-synthesis approach. Moreover, this is the first resynthesis algorithm that compares itself to common compilers like Qiskit [1] and TKET [23].

Since this approach transforms quantum circuits into quantum circuit, it is technically a transpilation algorithm rather than a compilation algorithm. We will use these terms interchangeably.

Note that the global nature of our proposed approach makes it probably unsuitable as an intermediate step in an existing compilation, unlike for example peep-hole-optimization.

Our algorithm is a combination of three previous algorithms. We will describe these algorithms in more detail in the next section, but for the familiar reader, we give a short overview here. We start with representing a given circuit in an intermediate representation (IR). We use the mixed ZX-phase polynomial IR that was used in [9] which is essentially a sequence of Paulistrings either over the alphabet $\{Z, I\}$ or $\{X, I\}$. This is a universal representation. We synthesize the circuit using the current state-of-the-art phase polynomial synthesis algorithm [24] and optimize the generated CNOTs with PermRowCol [17]. Then, we used simulated annealing to simplify the IR by adding a few CNOTs to the circuit. This last step essentially removes common CNOTs from the original Paulistrings.

Note that Qiskit now uses SABRE

Unfortunately, due to the heuristic nature of our algorithm, we cannot make claims about the minimality of newly generated circuits. However, by comparing our algorithm to Qiskit, TKET, and it's subcomponents, we know that the generated circuits have significantly less CNOTs than previous methods.

Our approach is very powerful for circuits generated from multi-qubit Pauli-interactions such as those generated by transforming a Hamiltonian to a quantum circuit.

In the remainder of this paper we will describe quantum circuit re-synthesis problem in II, then we describe our algorithm in III. We compare the performance of the proposed method in IV. We give an in-depth discussion of the assumptions we made in the design of this algorithm and how one can or

cannot deal with them (V). Lastly, we summarize and point to future research directions in VI.

II. Problem description

The problem that is addressed in this paper is commonly known as the "qubit routing problem" [5]. It is the problem that many quantum computers only support multi-qubit interactions between specific pairs of qubits. This means that not every arbitrary quantum program (quantum circuit) can be immediately executed on an arbitrary quantum computer. The quantum circuit will need to be changed first to accommodate these connectivity constraints (transpilation). Due to the "no-cloning theorem" [27], we are unable to reuse the existing classical methods because they require the copying of quantum states.

As the name suggests, the qubit routing problem is historically solved by routing the qubits: moving the qubits to different registers by applying SWAP operations until the connectivity constraints are satisfied [5]. However, each additional SWAP gate is generally implemented using 3 CNOT gates, which can generate significant overhead for the execution time of the quantum circuit. Since current state-of-the-art quantum computers are noisy, with low decoherence times, and sometimes sparsely connected qubits, the routing overhead can make a circuit impossible to execute.

Thanks to a recent paradigm shift, there is now a class of transpilation methods that do not rely on SWAP gates [20, 11, 8, 16, 17]. Instead, these algorithms rely on finding a scalable representation of the quantum circuit from which a completely new, but equivalent, circuit can be synthesized. During the synthesis process, only multiqubit interactions that are natively allowed are generated, hence the name "architecture-aware resynthesis" [16]. Most of these methods rely on the concept of "Steiner trees" to find how the existing connections can be optimally used. As such, this class of transpilation methods is sometimes called "Steiner-tree-based methods" [17]. An added benefit of resynthesizing the quantum circuit is that it will also be immediately optimized.

A. Mixed ZX-phase polynomial intermediate representation

Our method relies on the intermediate representation named the "mixed ZX-phase polynomial" [9]. We describe it here in detail and how it can be used for resynthesis. Unlike most existing literature, we will not rely on ZX-calculus [26] for this. It is true that mixed ZX-phase polynomials have a very intuitive representation in ZX-calculus, but its use is not yet mainstream enough to assume the reader's familiarity.

Given an arbitrary quantum circuit, we can rewrite that circuit into a sequence of multi-qubit Pauli operations $e^{i\alpha \otimes XYZI}$ called "Paulistrings". These operations are usually generated when transforming a Hamiltonian into a quantum circuit, and we assume the reader's familiarity with these operations. Note that a single-qubit gate can also be described as a Paulistring where every letter in the string is I except for one, which corresponds to the axis of rotation for that single-qubit gate.

A circuit consisting only of Paulistrings can then be rewritten into a sequence of Paulistrings either over the alphabet $\{Z, I\}$ (Z phase gadget) or $\{X, I\}$ (X phase gadget). Since Z-phase gadgets mutually commute and X-phase gadgets also, we can group the sequence into sequences of mutually commuting phase gadgets. A sequence of Z-phase gadgets is called a "phase polynomial" [3, 4], hence the name "mixed ZX-phase polynomial".

The advantage of the use of Paulistrings is that they can be efficiently represented. It is fully characterized by the string itself and its angle of rotation α. For the Z- and X-phase gadgets, the string can be represented as a bitstring (given that you know it is a Z- or X-phase gadget). Thus the original quantum circuit can be represented as matrix of $(n+2)\times m$, where n is the number of qubits ($+1$ for the angle, $+1$ for whether it is a Z- or X-phase gadget), and m is the number of phase gadgets in the mixed ZX-phase polynomial. Additionally, this representation behaves nicely when CNOTs are acted upon it.

Consider a circuit in which our intermediate representation is a black box. If two consecutive CNOTs were applied at the start of the circuit, the circuit would remain semantically the same. If one were to then move one of the CNOTs through the black box, the intermediate representation changes as follows:

- For **Z-phase gadgets**, the **target** qubit is added to the **control** qubit (modulo 2) in the bitstring.
- For **X-phase gadgets**, the **control** qubit is added to the **target** qubit (modulo 2) in the bitstring.

And the CNOT that was "pushed through" is now behind the black box.

If, at any point during this process, there is a single qubit phase gadget in the *first* sequence of mutually commuting phase gadgets, that phase gadget can be removed from the intermediate representation, and added as a single qubit gate just before the black box.

See Figure 1 for a visual representation of this process. The multi-qubit gate represents the black box with the mixed ZX-phase polynomial representation inside it.

The key observation for this approach to resynthesis is that we need to pick CNOTs that are allowed by the architecture AND optimally reduce the complexity of the intermediate representation in the black box. We want to take advantage of the CNOT being pushed through when it changes the bitstring representations of each phase gadget such that it requires fewer CNOTs later.

In the next section, we will describe how we did this in more detail.

III. The algorithm

Once the circuit is in the mixed ZX-phase polynomial representation, we need an approximately optimal strategy to synthesize the circuit from it. In this paper, we have have combined three previously proposed algorithms [9, 17, 24] into a single procedure. The aim is that these three algorithms work together in such a way that the whole performs better than the separate parts.

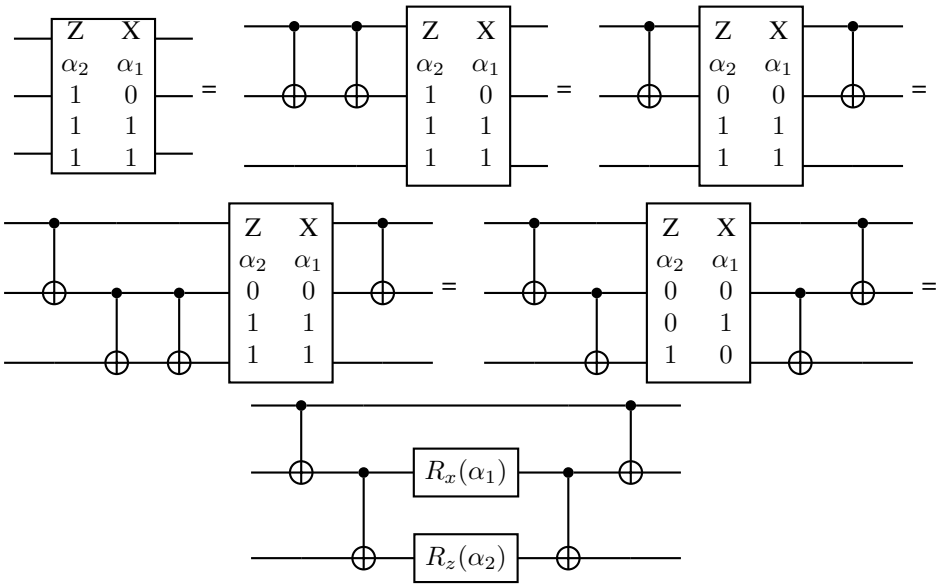

Fig. 1: Example of how the mixed ZX-phase polynomial intermediate representation can be used to generate a quantum circuit.

First, we use the simulated annealing approach proposed by [9]. The original algorithm adds pairs of CNOTs to the front of the circuit and pushes one of them through the intermediate representation, as explained in the previous section. The simulated annealing algorithm [21] is used to carefully add a few CNOTs to the circuit and simplify the intermediate representation. However, for the final synthesis of the intermediate representation, they use a naive synthesis strategy.

Thus, for this paper, we replace the naive synthesis strategy with a smarter one, namely the algorithm from [24]. We will call this algorithm "paritysynth" since it tries to synthesize the circuit in an optimal parity ordering for the given connectivity graph. If a subsequence of phase gadgets mutually commutes (because they are of the same type), we can synthesize them in arbitrary order. Thus, we should find the order that is optimal for the CNOT count. This is what paritysynth does.

Lastly, once the intermediate representation in the circuit is empty, we are left with a long sequence of accumulated CNOTs after the black box. These CNOTs can then be optimized using the third algorithm: PermRowCol [17].

We will give a more in depth explanation of these algorithms in a future version of this paper. For now, we refer the interested reader to the original papers.

Add implementation details of these algorithms

explain Steiner trees [14]

The novelty of our work is that these algorithms have not yet been used in unison before nor compared against common transpilers like Qiskit and TKET. In the next section, we will see that this combination performs much better than the algorithms on their own.

IV. Results

Update the text to fit the new results in the tables.

To evaluate our transpilation procedure, we generated 100 random circuits containing m phase gadgets each acting on at least $\sqrt{n}$ (rounded) qubits, where n is the number of qubits.

The code generating the circuits and implementing our transpilation strategy can be found on Github [1]. Our results are generated using the Jupyter notebook "notebooks/6. Phase Circuit to Quantum Circuit.ipynb".

We compare our algorithm against our own implementation of the ParitySynth algorithm, the original implementation of the simulated annealing approach, the Qiskit transpiler, and the TKET compiler. For Qiskit, we use the normal "transpile()" method with optimization level 3. For TKET, we have recreated the default compilation pass with optimization level 2 from documentation [2]. For the simulated annealing we use 100 iterations and 5 CNOT blocks.

As target devices, we used common connectivity constraints of IBM quantum computers. The graphs representing these devices can be found in Figure 2. Note that the results for Valencia-like graphs are currently missing and will be added for the next version of this paper. We chose these devices to give an idea of the scalability of our approach and the effect of different real-world graphs on performance. We encourage readers who are interested in the performance for other graphs to adjust the provided Jupyter notebook for their use cases.

We show the average CNOT counts for our randomly generated circuits on the different connectivity graphs: 20-qubit devices in Figure **??**, a 14-qubit device in Figure **??**, and a 5-qubit device in Figure **??**. We show the average CNOT count for the originally generated, naively decomposed circuit

[1] https://github.com/Aerylia/pauliopt

[2] https://cqcl.github.io/pytket-qiskit/api/index.html#default-compilation

Gadgets	Uncompiled	Qiskit [1]	TKET [23]	SG [16]	Par [24]	An [9]	SG+RT	SG+RT→An	An+SG+RT	Par+RT→An
1	5.7	5.46	**3.92**	5.7	*5.96*	5.7	5.44	4.87	4.31	4.81
2	12.1	10.68	**7.93**	11.69	*12.78*	10.8	11.56	10.6	8.8	10.22
5	27.06	23.63	**15.57**	24.22	*25.47*	24.0	23.37	20.46	18.21	20.23
10	50.7	*43.09*	**27.58**	38.01	38.95	47.2	35.21	32.07	29.56	31.87
50	242.44	*199.98*	121.59	108.25	106.11	213.48	104.61	97.45	93.01	**91.33**
100	477.74	*392.98*	239.91	188.02	187.99	419.1	184.17	175.76	169.94	**168.65**

(a) 5-qubit Valencia-like devices

Gadgets	Uncompiled	Qiskit [1]	TKET [23]	SG [16]	Par [24]	An [9]	SG+RT	SG+RT→An	An+SG+RT	Par+RT→An
1	5.44	5.16	**4.6**	5.44	5.44	5.46	5.36	*5.57*	4.98	5.46
2	10.28	9.08	**7.12**	*9.88*	9.39	9.22	9.63	9.07	7.69	9.0
5	24.06	*21.02*	**14.71**	20.41	18.76	19.82	19.16	17.76	15.51	16.3
10	43.98	37.11	25.2	32.32	29.33	*37.66*	29.47	27.32	24.33	**24.23**
50	202.26	*170.64*	112.84	90.0	78.0	155.62	85.5	79.85	75.31	**67.62**
100	416.54	*351.66*	230.88	163.6	139.04	306.76	159.08	149.98	143.47	**125.21**

(b) 5-qubit Yorktown-like devices

Gadgets	Uncompiled	Qiskit [1]	TKET [23]	SG [16]	Par [24]	An [9]	SG+RT	SG+RT→An	An+SG+RT	Par+RT→An
1	19.68	19.35	**14.93**	20.04	*20.42*	20.36	19.8	19.92	18.16	20.11
2	38.8	37.27	**29.09**	38.78	*39.8*	38.6	38.66	37.67	35.2	37.73
5	94.44	88.85	**63.94**	*93.12*	93.02	89.94	93.1	88.05	82.53	86.13
10	183.78	172.09	**123.71**	175.78	174.92	*177.18*	175.64	166.7	158.03	162.0
50	919.24	849.91	622.18	623.2	609.45	*850.76*	598.46	579.1	557.07	**555.07**
100	1833.14	*1690.4*	1223.35	1061.16	1018.56	1688.56	1041.18	1008.54	981.74	**960.73**

(c) 14-qubit Melbourne-like devices

Gadgets	Uncompiled	Qiskit [1]	TKET [23]	SG [16]	Par [24]	An [9]	SG+RT	SG+RT→An	An+SG+RT	Par+RT→An
1	27.1	26.81	**21.33**	*28.9*	27.94	28.1	28.51	28.1	25.55	27.16
2	55.16	53.74	**38.37**	*59.0*	58.05	55.56	58.71	56.25	51.66	54.97
5	137.88	132.82	**92.71**	*146.14*	141.68	134.64	146.09	139.27	129.38	131.21
10	278.36	269.1	**184.88**	*295.36*	273.88	268.84	*295.36*	280.4	264.57	258.88
50	1366.58	*1311.36*	**946.56**	1169.5	1095.4	1310.86	1144.14	1100.55	1072.16	1030.03
100	2730.04	*2619.86*	1919.77	1914.16	1780.62	2616.5	1893.95	1842.18	1806.01	**1705.68**

(d) 20-qubit Johannesburg-like devices

Gadgets	Uncompiled	Qiskit [1]	TKET [23]	SG [16]	Par [24]	An [9]	SG+RT	SG+RT→An	An+SG+RT	Par+RT→An
1	27.12	26.8	**20.63**	27.32	*29.04*	27.94	26.81	26.95	24.43	27.34
2	53.46	52.0	**36.1**	53.64	*57.33*	52.94	53.34	51.98	48.72	53.96
5	132.88	126.11	**90.57**	134.7	*141.82*	130.84	134.69	128.12	120.69	131.55
10	270.46	256.31	**180.1**	267.18	*279.74*	261.24	267.18	251.17	240.36	255.55
50	1384.62	1295.42	**938.44**	1090.79	1095.16	*1313.02*	1069.41	1025.26	997.01	1021.67
100	2733.66	2561.52	1904.46	1773.23	1763.2	*2605.04*	1750.49	1693.71	1656.43	**1681.05**

(e) 20-qubit Singapore-like devices

TABLE I: Average CNOT counts over 100 random circuits with different numbers of random phase gadgets and compiled for different devices.
Uncompiled is the CNOT count before compilation. Qiskit [1] and TKET [23] represent CNOT count after compiling using Qiskit and TKET, respectively. GS [16], Par [16], and An [9] represent the three algorithms on which our work is based and added as a baseline.
The last four columns are our proposed methods. SG+RT is the GS [16] method with added Reverse Traversal [12]. SG+RT→An is SG+RT followed by 1 pass of the annealer. An+SG+RT is the annealer where at every iteration the SG+RT algorithm is used for calculating the CNOT count. Lastly, Par+RT→An is the same as SG+RT→An but using the algorithm from Par [24]. Numbers in bold are the smallest CNOT counts in the row, and italicized numbers are the highest. Smaller counts are better.

(original), Qiskit, TKET, decomposed using only paritysynth (parity), naively decomposed and optimized by the annealer (annealer), and our combined approach (annealer+parity). Note that the results for our approach are not yet reallocating the qubits. A later version of this paper will have that, and the reallocation will reduce the CNOT count.

It is clear that the addition of the annealer improves the paritysynth. For larger amounts of phase gadgets (≥ 50), the addition of paritysynth to the annealer also improves it.

We note that TKET is outperforming Qiskit, so we will focus our analysis for existing compilers to TKET's performance. TKET closely follows the performance of the annealer. If the circuit has many phase gadgets ($\geq$ 50, which is expected), our method does improve the CNOT count quite significantly with up to 50% for circuits with 100 phase gadgets.

V. Discussion

The proposed approach to the transpilation of quantum circuits relies on a few assumptions. In this section, we list these and discuss why we do not expect them to be a problem.

Gadgets	Qiskit [1]	TKET [23]	SG [16]	Par [24]	An [9]	SG+RT	SG+RT→An	An+SG+RT	Par+RT→An
1	0.013	**0.0858**	0.001	*0.0018*	0.0067	0.0915	0.0407	0.317	0.0522
2	0.025	**0.1816**	0.0016	*0.0032*	0.0086	0.1615	0.0634	0.4634	0.0888
5	0.0466	**0.3519**	0.0029	*0.0054*	0.0114	0.2843	0.0964	0.6744	0.1465
10	*0.0899*	**0.6232**	0.004	0.0085	0.0149	0.4031	0.1315	0.9087	0.2256
50	*0.4352*	2.6337	0.0107	0.0311	0.0365	1.2291	0.3201	2.012	**0.7917**
100	*0.8999*	5.2591	0.0215	0.071	0.063	2.4885	0.6062	3.6014	**1.7468**

(a) 5-qubit Valencia-like devices

Gadgets	Qiskit [1]	TKET [23]	SG [16]	Par [24]	An [9]	SG+RT	SG+RT→An	An+SG+RT	Par+RT→An
1	0.0124	**0.096**	0.0009	0.0018	0.0069	0.089	*0.038*	0.2951	0.0499
2	0.0211	**0.1643**	*0.0017*	0.0029	0.0088	0.153	0.0592	0.4389	0.0804
5	*0.0442*	**0.3377**	0.0028	0.0053	0.0123	0.2633	0.0924	0.6606	0.1373
10	0.0753	0.5666	0.0039	0.0079	*0.0157*	0.3879	0.1282	0.8687	**0.207**
50	*0.3587*	2.5011	0.0101	0.0285	0.0374	1.1388	0.3002	1.9016	**0.7144**
100	*0.7246*	5.0354	0.0201	0.0604	0.0618	2.2985	0.5547	3.266	**1.5148**

(b) 5-qubit Yorktown-like devices

Gadgets	Qiskit [1]	TKET [23]	SG [16]	Par [24]	An [9]	SG+RT	SG+RT→An	An+SG+RT	Par+RT→An
1	0.0413	**0.3135**	0.006	*0.0295*	0.0098	0.5552	0.219	1.6737	0.6791
2	0.0804	**0.6688**	0.0116	*0.0532*	0.015	1.0415	0.3836	2.8372	1.2681
5	0.1779	**1.3963**	*0.0224*	0.0971	0.0273	1.9875	0.6941	4.9686	2.3564
10	0.3516	**2.6953**	0.0326	0.1566	*0.0416*	3.2337	1.0666	7.4732	4.0009
50	2.3098	14.5915	0.102	0.7816	*0.1778*	9.8345	2.7383	17.9599	**17.6075**
100	*4.7391*	27.1506	0.1769	1.428	0.2812	18.8626	4.7366	28.4287	**34.4294**

(c) 14-qubit Melbourne-like devices

Gadgets	Qiskit [1]	TKET [23]	SG [16]	Par [24]	An [9]	SG+RT	SG+RT→An	An+SG+RT	Par+RT→An
1	0.0532	**0.4567**	*0.0117*	0.0839	0.0133	1.1052	0.4321	3.2735	1.8964
2	0.1188	**0.9215**	*0.0243*	0.1434	0.0236	2.1109	0.7967	5.8083	3.454
5	0.2735	**2.0804**	*0.0521*	0.3152	0.0463	4.4786	1.5671	11.3152	7.2496
10	0.5497	**4.3389**	*0.0857*	0.537	0.0883	*7.7223*	2.6226	18.5605	12.8408
50	*3.2907*	**23.1061**	0.2146	2.5235	0.3594	20.9288	5.9286	38.7839	58.1048
100	*6.6193*	46.3972	0.3768	4.9302	0.6347	39.091	10.0422	61.1307	**117.7887**

(d) 20-qubit Johannesburg-like devices

Gadgets	Qiskit [1]	TKET [23]	SG [16]	Par [24]	An [9]	SG+RT	SG+RT→An	An+SG+RT	Par+RT→An
1	0.0546	**0.4602**	0.011	*0.0803*	0.0131	1.0679	0.4066	3.0566	1.8394
2	0.1116	**0.858**	0.0222	*0.137*	0.0217	1.8677	0.6889	5.0574	3.184
5	0.2696	**2.0595**	0.0482	*0.283*	0.0459	4.3044	1.4979	10.5482	6.699
10	0.5531	**4.093**	0.0802	*0.5164*	0.08	7.3885	2.4445	17.2524	12.29
50	3.8567	**21.638**	0.2022	2.4298	*0.3596*	19.9057	5.654	37.3759	56.0521
100	8.1115	47.5262	0.3965	5.2028	*0.6972*	38.9048	9.9294	61.696	**116.6315**

(e) 20-qubit Singapore-like devices

TABLE II: Average runtime in seconds over 100 random circuits with different numbers of random phase gadgets and compiled for different devices. The column naming is consistent with Table I.
Bold numbers correspond to the method with the lowest CNOT count in the row and italicized numbers the highest. This is done to more easily see the runtime trade-off for better CNOT counts. Lower runtimes are better.

A. Original circuit quality

Our transpilation approach assumes that the original creator of the quantum circuit to be transpiled is not familiar with quantum circuit optimization and the qubit routing problem. Therefore, we assume that the given circuit is poorly optimized. This is a sensible assumption because current use cases for quantum computers is the simulation of quantum processes. Thus, the user of quantum computers are physicists and chemists, who might not be familiar with quantum computer science.

As such, we expect that most NISQ programs will be generated from a given Hamiltonian. During this process, the Hamiltonian is transformed into a sequence of Paulistrings, which make the mixed ZX-phase polynomial a native intermediate representation to the original program. In fact, using Paulistrings as an intermediate representation was already proposed for the Paulihedral compiler [13].

We will mention that if the original quantum circuit is already heavily optimized for the target hardware, we do not expect our transpilation method to still improve the circuit. As we already could see in the results for simple circuits with few phase gadgets. The remainder of the paper will discuss why this could be the case and how one could deal with that.

B. The native multi-qubit interactions of the quantum computer

The only multi-qubit interaction that is generated by our algorithm is the CNOT gate. This makes sense from the point of view that together with arbitrary single qubit gates, this makes a universal gate set. However, the physical implementation of native multiqubit interactions on current quantum hardware is generally not a CNOT. The devices can execute a CNOT using

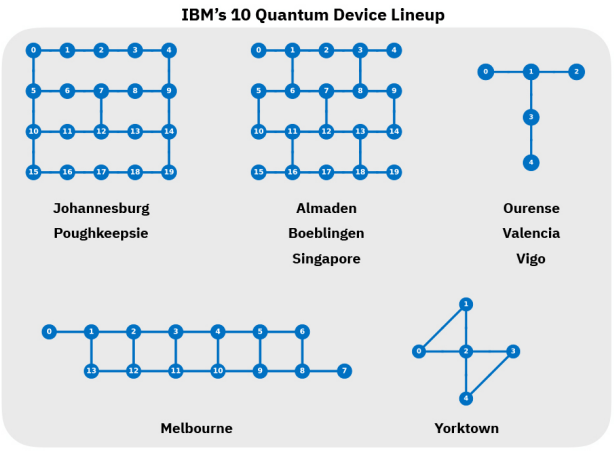

Fig. 2: Graphs representing the connectivity constraints for different IBM quantum computers. Picture taken from https://www.ibm.com/blogs/research/2019/09/quantum-computation-center/.

their native interaction and some single qubit gates, but it is not immediately a CNOT. Some compilation is still necessary!

In general, it takes one of these multiqubit interactions to simulate the CNOT, making it seem that this does not affect the number of multiqubit interactions. However, some devices (e.g. ion-traps) allow multi-qubit interactions between all qubits. This would remove the need to reduce the phase gadgets into smaller interactions.

But let's assume, for the sake of argument, that we have a device with a native 2-qubit interaction. In that case, we can still simulate the CNOT using some single-qubit gates. However, on some devices these interactions will be implemented in an "echoed" form to improve environmental noise [6], doubling all 2-qubit gates. Moreover, in cases where a 2-qubit phase gadget acts on two connected qubits on the topology, the phase gadget might be turned into a native 2-qubit interaction directly (using single-qubit gates) without the need for generating CNOTs.

Transpiling a phase gadget directly to the native 2-qubit gate would require that 2-qubit gate to be tunable with the angle of the phase gadget. Some devices allow this (e.g. some IBM devices [6]), in which case they need to be calibrated for all possible angles rather than just the one angle needed for the CNOT (or two if it is echoed). But calibration of many angles is more difficult. So herein lies a trade-off: implement a 2-qubit phase gadget with 2 "good" interactions (from CNOTs) or 1 "bad" interaction (directly native)?

Our transpilation procedure will change the balance of this trade-off. Due to the resynthesis procedure, the second interaction of the decomposed phase gadget is pushed to the end of the circuit and might be optimized away. On the other hand, if the second interaction isn't optimized away, we might want to use the native interaction. But when the second interaction isn't pushed through the later phase gadget, it will change the entire circuit to the point that it is not obvious which option is better. What is clear in this trade-off is that, because the second interaction might be optimized away, the quality of the tunable 2-qubit interaction needs to be better than twice the fixed-angle interaction.

Additionally, our transpilation approach is not suitable for a look-up-and-replace strategy to introduce the tunable 2-qubit interactions, because it probably will not generate CNOT-(single qubit gate)-CNOT sequences. Thus, in case tunable interactions are allowed, the synthesis approach will need to take that into account from the start.

C. The native single qubit gates

The only single-qubit gates that our transpilation approach generates are X and Z rotations. The assumption here is that these are equally desirable for the hardware, but this is not necessarily true. For example, in superconducting devices, rotations around the Z axis are "virtual". They do not change the qubit itself but the equipment around it (i.e. it resets the clock). This means that Z rotations are essentially noiseless while X rotations require noise manipulations of the qubit. Thus, it is beneficial to generate more Z rotations that X rotations.

Given that X rotations will have to be calibrated and Z rotations do not, it is better to calibrate the X rotations for one or two angles and simulate all arbitrary X rotations $R_X(\alpha)$ as $HR_Z(\alpha)H$, where the H gate is created from the calibrated X rotations and the virtual Z rotation.

This adjustment can be immediately added to our proposed algorithm, but we can also make a slight adjustment to our algorithm that might influence the design of future quantum hardware. During the synthesis procedure, we alternate between decomposing sequences of Z-phase gadgets and X-phase gadgets. What we can do is to generate a H gate on every qubit before the sequence of X-phase gadgets and after the sequence of X-phase gadgets. Due to the connectivity constraints, we do no know which qubits are needed to synthesize the X-phase gadgets, so we need to put the H gate on all qubits (unnecessary H gates can be removed after synthesis). Now, the sequence of X-phase gadgets is turned into a sequence of Z-phase gadgets, and we can continue the algorithm like before, except that when a generated CNOT is pushed through the barrier of H gates, its direction is reversed. In case that the sequence of X-phase gadgets is longer than the number of qubits, this reduces the total amount of H gates required.

The reason we believe strategy might influence the design of future quantum hardware is that placing a H gate on all qubits is a very common operation in quantum algorithms [7, 22, 10]. If the transpilation also generates these frequently, it might be sensible to make this operation "hardware accelerated". I.e. make some hardware adjustments so that this specific thing can be done better/faster/easier.

D. Effect of gate fidelity

Another effect that our algorithm does not natively take into account is the difference in gate fidelity. On current quantum

computers, not every qubit can be equally well controlled and not every multi-qubit interaction has the same noise. Our results do not take this into account and assume that all gates are equally good. We have discussed some of these aspects when discussing the implementation of the natively supported gates by the quantum devices, but we will discuss the effect of fidelity differences in a broader sense here.

Although we assume for our results that all gates are equally good, our algorithm can be adjusted to take gate quality into account. The most straightforward way to do this is to make the connectivity graph a weighted graph where the weights correspond to the gate fidelities. More generally, we can redefine the measure of "qubit distance" when calculating the minimal Steiner trees. Within our implementation, this would mean to redefine the value of the shortest path between every qubit in the topology. Since we approximate the Steiner-tree by calculating the minimal spanning tree over the all-pairs shortest paths. This measure of qubit distance can then take more into account than only the 2-qubit gate fidelity, but also the quality of the qubit itself. Similarly, we can use gate fidelity as a tie-breaker when choosing CNOTs or qubits in our algorithm.

In general, we need to find better strategies to take more dynamic sources of noise into account. The algorithm can be adjusted to reduce crosstalk between qubits, improve gate scheduling, or take into account other device-specific sources of error. This will require more collaboration between hardware manufacturers and transpiler designers, as well as availability of more detailed specifications for the available hardware.

E. Effect of approximate Steiner-trees

Lastly, we discuss the elephant in the room: finding a minimal Steiner tree is an NP-hard problem [14]. The only reason our algorithm is performant is that we use a polynomial approximation. This means that the paths that we find between qubits might not be optimal and we could reduce the CNOT count even further.

It is possible that the types of connectivity graphs on quantum computers are sparse enough so that the approximation is able to find the minimum Steiner tree, but we do not know.

In case the approximate Steiner trees are insufficient, we could still find a minimal Steiner tree using quantum computers. Thanks to adiabetic quantum computing, we should be able to find minimal Steiner trees with quantum annealers [15]. In theory, we could also use gate-based quantum computers to do this, but we run into a chicken-egg problem. Because current quantum annealers have more qubits to their disposal and they do not need a transpiled quantum circuit, we can actually use quantum computers to compile quantum circuits. In fact, this has recently already been done in the ISAAQ compiler [19].

More generally, we might be able to describe the full parity network problem (solved in paritysynth) in a fashion suitable for quantum annealers. There is a lot of interaction between generating which parity on which qubit in which order, that might be very suitable for a quantum annealing formulation.

VI. Conclusion and Future work

In this paper, we proposed a full-stack quantum circuit resynthesis approach to transpilation. We compared it to similar state-of-the-art algorithms as well as the Qiskit transpiler and TKET. We showed that for more complicated circuits our proposed approach improves the CNOT count.

We argue that in transpilation, we need to take a more holistic approach that takes the semantics of the circuit into account, not just the gates themselves. And we believe that the improved CNOT counts that we have shown support that argument.

However, our algorithm can still be improved. The fact that the simulated annealing is improving the synthesized circuit shows that our heuristics are non-optimal. For now, this is good enough, but we should be able to do better.

Additionally, we need to generalize our algorithm to have Paulistrings as our intermediate representation, just like the Paulihedral compiler [13]. Transforming the Paulistrings into a mixed ZX-phase polynomial can generate many extra phase gadgets which is inefficient.

Lastly, one of the strengths of compiling methods like Qiskit and TKET is that they are able to move the qubits on the architecture. The implementation for the current results does not choose an optimal mapping of the qubit, nor does it move the qubits around. This means that improvements can still be made, as shown in [17]. The next version of this paper will include the dynamic reallocation from PermRowCol [17] and hopefully the reverse traversal strategy from [12] to find an optimal mapping if time allows. Regardless, applying these methods will only further improve the results.

Acknowledgement

I would like to thank Richie Yeung and Stefano Gogioso for helpful discussions and insights into their code base on which the full algorithm was built.

References

[1] A-tA-v et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2021. DOI: 10.5281/zenodo.2573505.

[2] Amazon Web Services. *Amazon Braket*. 2023. URL: https://aws.amazon.com/braket/.

[3] Matthew Amy, Parsiad Azimzadeh, and Michele Mosca. "On the controlled-NOT complexity of controlled-NOT–phase circuits". en. In: *Quantum Science and Technology* 4.1 (Sept. 2018), p. 015002. ISSN: 2058-9565. DOI: 10.1088/2058-9565/aad8ca.

[4] Matthew Amy, Dmitri Maslov, and Michele Mosca. "Polynomial-Time T-Depth Optimization of Clifford+T Circuits Via Matroid Partitioning". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.10 (Oct. 2014), pp. 1476–1489. ISSN: 1937-4151. DOI: 10.1109/TCAD.2014.2341953.

[5] Alexander Cowtan et al. “On the Qubit Routing Problem”. en. In: (2019), 32 pages. DOI: 10.4230/LIPICS.TQC.2019.5.

[6] Nathan Earnest, Caroline Tornow, and Daniel J. Egger. “Pulse-Efficient Circuit Transpilation for Quantum Applications on Cross-Resonance-Based Hardware”. In: *Physical Review Research* 3.4 (Oct. 2021), p. 043088. ISSN: 2643-1564. DOI: 10.1103/PhysRevResearch.3.043088. URL: https://link.aps.org/doi/10.1103/PhysRevResearch.3.043088 (visited on 10/03/2022).

[7] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. “A Quantum Approximate Optimization Algorithm”. In: (2014). DOI: 10.48550/ARXIV.1411.4028. URL: https://arxiv.org/abs/1411.4028.

[8] Vlad Gheorghiu et al. “Reducing the CNOT Count for Clifford+T Circuits on NISQ Architectures”. In: (2020). DOI: 10.48550/ARXIV.2011.12191. URL: https://arxiv.org/abs/2011.12191 (visited on 10/03/2022).

[9] Stefano Gogioso and Richie Yeung. “Annealing Optimisation of Mixed ZX Phase Circuits”. In: (2022). DOI: 10.48550/ARXIV.2206.11839. URL: https://arxiv.org/abs/2206.11839 (visited on 10/03/2022).

[10] Lov K. Grover. “A fast quantum mechanical algorithm for database search”. en. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC ’96*. Philadelphia, Pennsylvania, United States: ACM Press, 1996, pp. 212–219. ISBN: 978-0-89791-785-8. DOI: 10.1145/237814.237866. URL: http://portal.acm.org/citation.cfm?doid=237814.237866.

[11] Aleks Kissinger and Arianne Meijer van de Griend. “CNOT Circuit Extraction for Topologically-Constrained Quantum Memories”. In: *Quantum Information and Computation* 20.7&8 (June 2020), pp. 581–596. ISSN: 15337146, 15337146. DOI: 10.26421/QIC20.7-8-4. URL: http://www.rintonpress.com/journals/doi/QIC20.7-8-4.html (visited on 10/03/2022).

[12] Gushu Li, Yufei Ding, and Yuan Xie. “Tackling the qubit mapping problem for NISQ-era quantum devices”. In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2019, pp. 1001–1014.

[13] Gushu Li et al. “Paulihedral: A Generalized Block-Wise Compiler Optimization Framework for Quantum Simulation Kernels”. In: *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. Lausanne Switzerland: ACM, Feb. 2022, pp. 554–569. ISBN: 978-1-4503-9205-1. DOI: 10.1145/3503222.3507715. URL: https://dl.acm.org/doi/10.1145/3503222.3507715 (visited on 10/03/2022).

[14] Ivana Ljubić. “Solving Steiner trees: Recent advances, challenges, and perspectives”. en. In: *Networks* 77.2 (Mar. 2021), pp. 177–204. ISSN: 0028-3045, 1097-0037. DOI: 10.1002/net.22005.

[15] Andrew Lucas. “Ising formulations of many NP problems”. In: *Frontiers in Physics* 2 (2014). ISSN: 2296-424X. DOI: 10.3389/fphy.2014.00005. URL: http://journal.frontiersin.org/article/10.3389/fphy.2014.00005/abstract.

[16] Adriana Meijer - van de Griend and Ross Duncan. “Architecture-Aware Synthesis of Phase Polynomials for NISQ Devices”. In: (2020). DOI: 10.48550/ARXIV.2004.06052. URL: https://arxiv.org/abs/2004.06052 (visited on 10/03/2022).

[17] Adriana Meijer - van de Griend and Sarah Meng Li. “Dynamic Qubit Allocation and Routing for Constrained Topologies by CNOT Circuit Re-Synthesis”. In: (2022). DOI: 10.48550/ARXIV.2205.00724. URL: https://arxiv.org/abs/2205.00724 (visited on 10/03/2022).

[18] Prakash Murali et al. “Formal constraint-based compilation for noisy intermediate-scale quantum systems”. en. In: *Microprocessors and Microsystems* 66 (Apr. 2019), pp. 102–112. ISSN: 01419331. DOI: 10.1016/j.micpro.2019.02.005.

[19] Soshun Naito et al. “ISAAQ: Ising Machine Assisted Quantum Compiler”. In: *arXiv preprint arXiv:2303.02830* (2023).

[20] Beatrice Nash, Vlad Gheorghiu, and Michele Mosca. “Quantum circuit optimizations for NISQ architectures”. In: *Quantum Science and Technology* 5.2 (2020), p. 025010.

[21] Martin Pincus. “A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems”. en. In: *Operations Research* 18.6 (Dec. 1970), pp. 1225–1228. ISSN: 0030-364X, 1526-5463. DOI: 10.1287/opre.18.6.1225.

[22] P.W. Shor. “Algorithms for quantum computation: discrete logarithms and factoring”. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. Santa Fe, NM, USA: IEEE Comput. Soc. Press, 1994, pp. 124–134. ISBN: 978-0-8186-6580-6. DOI: 10.1109/SFCS.1994.365700. URL: http://ieeexplore.ieee.org/document/365700/.

[23] Seyon Sivarajah et al. “t—ket⟩: a retargetable compiler for NISQ devices”. In: 6 (Jan. 2021), p. 014003. ISSN: 2058-9565. DOI: 10.1088/2058-9565/ab8e92.

[24] Vivien Vandaele, Simon Martiel, and Timothée Goubault de Brugière. “Phase Polynomials Synthesis Algorithms for NISQ Architectures and Beyond”. In: *Quantum Science and Technology* 7.4 (Oct. 2022), p. 045027. ISSN: 2058-9565. DOI: 10.1088/2058-9565/ac5a0e. URL: https://iopscience.iop.org/article/10.1088/2058-9565/ac5a0e (visited on 09/29/2022).

[25] Davide Venturelli et al. “Compiling quantum circuits to realistic hardware architectures using temporal planners”. In: *Quantum Science and Technology* 3.2 (Apr. 2018), p. 025004. ISSN: 2058-9565. DOI: 10.1088/2058-9565/aaa331.

[26] John van de Wetering. “ZX-calculus for the working quantum computer scientist”. In: *arXiv preprint arXiv:2012.13966* (2020).

[27] William K Wootters and Wojciech H Zurek. “A single quantum cannot be cloned”. In: *Nature* 299 (1982), pp. 802–803.